Ontological Design Patterns for the Management of Molecular Biological Knowledge

Jacqueline Renée Reich
University of Manchester, Department of Computer Science,
Oxford Road, Manchester M13 9PL, England
reich@cs.man.ac.uk

Ontology and Knowledge Management

Molecular biological knowledge has to be represented and managed in such a way that it can be disseminated and applied as an information resource within molecular biology or related areas, such as pharmacology and genetics. Molecular biologists interpret their scientific data by comparing new data against a large amount of existing knowledge in order to predict further conclusions. For instance, the DNA structure of an unknown DNA sequence, its function as a protein, and its evolutionary role are inferred from the degree of similarity with sequences of well-known proteins. The biological function of a protein is defined by both the detailed chemical properties of its surface and its structure, which together depend on the underlying amino acid sequence. One can "predict" function by recognising similarities of different types at the sequence level. The inference of function from protein sequence similarity searches is non-trivial and to be effective requires considerable biological knowledge. For example, to infer function from sequence or pattern database similarities, one must refer to the literature to find the necessary biological evidence to support that inference.

Various biology related research areas, such as medicine and molecular biology, started to build ontologies to represent parts of their knowledge and terminology, and to provide concise and non-ambiguous biological models (e.g. GALEN for medical language and knowledge (Rector et al. 1997)). Ontologies include concepts and instances of information, showing their relationships and possible constraints given by the knowledge area. A concept summarises features of its instances and names their generalisation. For instance, "flowers" becomes the name of the concept about various kinds of flowers, which is described by a definition and attributes about generalised features common to all its concrete instances. An attribute (e.g. leaf-colour) has a definition and additional semantic properties. The definitions include information about the context and well-formed use of the terms. Instances (e.g. the rose on my table) are described by values or objects, and belong to their concept.

The construction of an ontology for a molecular biology related theme, such as G-protein coupled receptors (GPCR), involves spatial, temporal, and functional limits and choices. For instance, part of a GPCR-ontology may define proteins involved in colour vision, such as red-, blue- and green-sensitive opsins. The instances of the various opsin concepts share general features or properties, such as an effector pathway. Within this pathway feature, the instance's version can be described how to activate or stimulate other proteins or enzymes. This GPCR-ontology may exclude the related group of bacteriorhodopsins, because these are light-sensitive proteins found in bacteria, but they do not interact with G-proteins. Bacteriorhodopsins are functionally unrelated to the colour vision opsins. The decision to integrate or exclude the bacteria-related proteins produces a dichotomy to accept or reject part of the GPCR terminology.

Ontological Design Patterns

Ontological Design Patterns (ODPs) provide a technique to increase the flexibility and reusability of biological ontologies. For example, the opsin ontology fragment could be extended to represent also rhodopsin without modifying the original ontology version. Opsins are involved in colour vision and found in cone photoreceptors. In contrast, rhodopsin detects dim light and is found in rod cells. The various opsins and rhodopsin have the same effector pathway, but different mutations in the proteins causes different kinds of blindness. The example shown below of the AddDynamicInfo ODP applied to the context of opsin and rhodopsin will show how ODPs can abstract and identify the design structures and semantic contexts of molecular biological phenomena in an incremental and iterative process. This example will illustrate the relatively new idea of developing and reusing design patterns to solve recurring ontology design problems.

The formalisation of ODPs ensures that an ontology can be built in a repetitive fountain-like approach including cycles of specification, classification and implementation. For instance, during a later design cycle, the original opsin ontology fragment could be definitely modified and linked with a rhodopsin-ontology. The capability to adjust and to improve the properties, elements, behaviour, structures, and tasks of ontologies isimportant as development and changes in on-going research and science continues.

ODPs can solve design problems of a context-specific ontology. The context of an ontology can change over

time, such as first excluding the bacteria related protein and focusing on human proteins but later integrating bacteriorhodopsin. Theoretically, the contexts can be related to processes and techniques for the creation, collection, indexing, organisation, distribution, access to and evaluation of scientific knowledge. For instance, the context of knowledge evaluation may move from a first goal to analyse scientific texts concentrating on opsin related literature to the goal to expand the analysis to rhodopsin related articles. To provide the necessary flexibility to broaden the range of analyses, the decomposition, organisation and classification of a scientific vocabulary have to consider the encapsulation, granularity, and dependency of concepts, and the performance, evolution, and reusability of the complete ontology. Temporal objectives, like our opsin-rhodopsin example, influence 1) how tightly sub-ontologies or individual concepts are designed and at which degree of granularity, 2) which concept-interfaces and inheritance-hierarchies are defined, and 3) which key-relationships are established among them.

So far, ten different ODPs are defined (Table 1) (Reich 1999a,b, in press), which can be implemented in various knowledge specification or programming languages.

TerminologicalHierarchy ODP	to compose concepts and instances into part-whole hierarchies
UpdateDependencies ODP	to notify interdependent concepts and instances if one concept is changed
DefinitionEncapsulation ODP	to define a family of definitions, and to encapsulate each one
ExpressionComposer ODP	same construction process for different concepts
ChainOfExpressions ODP	multiple concepts can fulfil a context
InteractionHider ODP	concept encapsulates interaction of concepts or instances
TerminologyOrganizer ODP	interface creates and organises concepts or instances, realised by sub-concepts
Mask ODP	to represent complete sub-concepts as instances
UnspecificTerm ODP	to manage unspecific instances at fine granularities
AddDynamicInfo ODP	to add information and meaning dynamically

Table 1. Ontological Design Patterns (ODPs)

AddDynamicInfo ODP

G-protein-coupled receptors (GPCR) are the largest family of cell-surface receptors. GPCRs can also be called G-protein-linked receptors. The GPCR family members have similar structures and are evolutionarily related. Our simple example involves opsins, the proteins for colour vision, and rhodopsin, the proteins for vision in dim light.

The AddDynamicInfo ODP dynamically adds information to instances, which can be withdrawn later. It provides a flexible alternative to sub-concepts for extending the functionality of an ontology. The AddDynamicInfo ODP can be used when an extension by sub-concepts is impractical. Sometimes a large number of independent extensions are possible and they would produce an explosion of sub-concepts to support every combination. In our example, the original ontology involves the representation of the information about opsins showing concepts, such as colour vision, cone cells, and disease-by-deficiency. To adapt the existing ontology to the information of rhodopsin, information about a different vision or cell type will be dynamically added. Generally, the integration of rhodopsin can be done in two different ways: 1) without using the AddDynamic ODP creating a new concept for each additional information, such as for dim light vision, and rod cells; or 2) using the AddDynamic ODP without modifying the core ontology based on the information of opsins.

The first approach requires more generalised and abstract concepts. The concepts of colour vision and dim light vision would be grouped into a higher concept named, for instance, "vision", and the concepts of rod cells and cone cells would be grouped into the concept "cell". The AddDynamicInfo ODP avoids those concepts high up in the ontological hierarchy. This ODP encloses the information to be added in another instance, which is called an `InfoAdder`. Any `InfoAdder` can alter or extend the original instance's functionality from the outside. The instance itself does not know anything about its `InfoAdders`. Instead of trying to support detailed features in an abstract constellation of concepts, a more general concept about GPCRs can be defined and information will be added incrementally by the `NameInfoAdder`, the `VisionInfoAdder`, or the `CellInfoAdder`. The `NameInfoAdder` will distinguish the expressions "G-protein-coupled receptors" and "G-protein-linked receptors", the `VisionInfo-Adder` the colour vision and the dim light vision, and the `CellInfoAdder` the context of cone or rod cells.

The following code shows how to implement `NameInfoAdder`, `CellInfoAdder` and `VisionInfoAdder`. We will assume that there is a concept called `Property-El`, which is an abstract concept for various properties and provides abstract operations, such as `Add`, `Remove` and `Change`.

```
class Property-El{
public:
    Element();
    virtual void Add();
    virtual void Remove();
    virtual void Change();
};
```

A sub-concept of Property-El is called an InfoAdder, which has more specific functions than Property-El. InfoAdder will have sub-concepts, such as NameInfoAdder, CellInfoAdder and VisionInfoAdder, which will be specific for various information changes. They can add operations for specific functionality. The important aspect of the AddDynamicInfo ODP is that it lets InfoAdders appear anywhere a Property-El can.

```
class InfoAdder : public Property-El{
public:
    InfoAdder(Element*);
    virtual void Add();
    virtual void Remove();
    virtual void Change();
private:
    Property-El* el;
};
```

InfoAdder adds information to the Property-El referenced by el. For each operation in the Property-El's interface, such as Add, InfoAdder defines a default implementation that passes the context on to el.

```
void InfoAdder::Add() {
el->Add();
}
```

Sub-concepts of InfoAdder define specific, added information. For instance, the NameInfoAdder adds a name element overriding the Add operation.

```
class NameInfoAdder: public InfoAdder {
public:
    NameInfoAdder (Element*, name-el);
    virtual void Add();
private:
    void addName(char*);
private:
    char* name-el; //Added name element
};

void NameInfoAdder::Add() {
    InfoAdder::Add();
    addName(name-el);
}
```

Experimental Results and Conclusions

Applying the AddDynamicInfo, two issues were found worthy of consideration: 1) An InfoAdder's interface, such as NameInfoAdder, has to conform to the interface of the original instance (e.g. the original version of the protein name, kind of vision or cell type), which will be a sub-concept of Property-El. 2) To ensure a conforming interface, original instances and InfoAdders should descend from a common concept, such as Property-El. This common concept should focus on defining an interface, not on storing data. The definition of the data representation should be deferred to sub-concepts, such as NameInfoAdder. Otherwise the concept of the original instance might become too complex, and the and sub-concepts will pay for features they do not need.

Tightly connected concepts are hard to reuse in isolation, since they depend on each other. This leads to monolithic ontologies where a concept can not be changed or removed without understanding and changing many other concepts. The application of the AddDynamicInfo ODP enables loose connections, which increase the probability that a concept can be reused by itself. This presupposes that an InfoAdder's interface conforms to the interface of the original instance, and that an InfoAdder changes the information of an instance from the outside covering it with an additional layer.

With AddDynamicInfo, ODP simple concepts can be defined and information added incrementally with InfoAdders. Information can be composed from simple pieces. It is easy to define new kinds of InfoAdders independently from the concepts of the instances they will extend even for unforeseen extensions.

All ODPs with code examples are available from the author. Contact: reich@cs.man.ac.uk.

Acknowledgements

This work was done at the Department of Computer Science of the University of Manchester. Many thanks to Daniel Buchholz for reviewing the manuscript.

References

Rector, A.L.; Bechhofer, S.; Goble, C. A.; Horrocks, I. R.; and Solomon W. D. 1997. The GALEN modelling language for medical terminology. *Artificial Intelligence in Medicine*, 9:139-171.

Reich, J. R. 1999. Ontological Design Patterns for the Integration of Molecular Biological Information. *Proceedings of the German Conference on Bioinformatics (GCB'99)*, 156-165, Hannover, Germany, http://www.bioinfo.de/isb/gcb99/ talks/reich

Reich, J. R. in press. Modelling Molecular Biological Information: Ontologies and Ontological Design Patterns: *Proceedings of the 11th Conference of the Information Resources Management Association (IRMA2000)*, Anchorage, Alaska